[Maven详解](https://www.cnblogs.com/hongwz/p/5456578.html)

**一.前言**

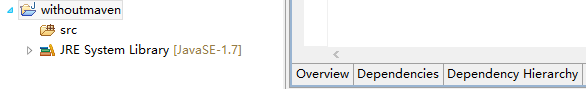
以前做过的项目中，没有真正的使用过Maven，只知道其名声很大，其作用是用来管理jar 包的。最近一段时间在项目过程中使用Maven，用Maven构建的web项目，其项目结构只停留在了解阶段，没有深入的使用与理解，刚好最近看了一篇关于Maven的详解；就开始深入学习一下Maven的具体应用。

**二.Maven的作用**

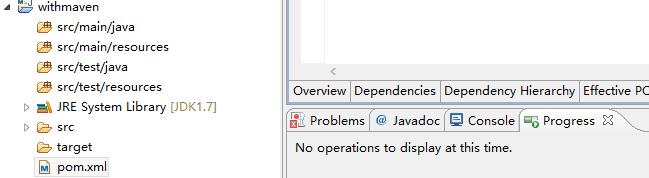
1. 在开发中，为了保证编译通过，我们会到处去寻找jar包，当编译通过了，运行的时候，却发现"ClassNotFoundException"，我们想到的是，难道还差jar包？
2. 每个Java项目的目录结构都没有一个统一的标准，配置文件到处都是，单元测试代码到底应该放在那里也没有一个权威的规范。
3. 因此，我们就要用到Maven（使用Ant也可以，不过编写Ant的xml脚本比较麻烦）----一个**项目管理工具**。
4. Maven主要做了两件事：

* 统一开发规范与工具
* 统一管理jar包

    下面我们来对比一下，首先建立一个普通的Java工程，是这样的：



   这个我们都很熟悉，src下建包写代码，那么配置文件放在哪里？单元测试代码放在哪里？没有一个统一标准，更多时候都是开发者的自由发挥，每个人有自己的风格，这并不十分适合团队协作。接下来，看一下使用maven构建一个普通Java项目之后的目录结构：



看到使用Maven构建的普通Java项目，对源代码、单元测试代码、资源乃至后续需要的文件都有专门的目录规划。

上面的最后有一个pom.xml，这是Maven的核心配置文件，pom称为Project Object Model（**项目对象模型**），它用于描述整个Maven项目，所以也称为Maven描述文件。

当然事情不会这么简单，接下来，继续进入Maven的世界吧。

**pom.xml**

打开pom.xml，最基础的是这样的：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <project xmlns="http://maven.apache.org/POM/4.0.0"      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">      <modelVersion>4.0.0</modelVersion>    <groupId>com.xrq.withmaven</groupId>    <artifactId>withmaven</artifactId>    <version>0.0.1-SNAPSHOT</version>    <build/>  </project> |

因为这个配置文件是Maven的核心，因此有必要详细解读一下pom.xml，来先看一下上面的几个：

**1、modelVersion**

　　指定了当前Maven模型的版本号，对于Maven2和Maven3来说，它只能是4.0.0

**2、groupId**

　　顾名思义，这个应该是公司名或是组织名。一般来说groupId是由三个部分组成，每个部分之间以"."分隔，第一部分是项目用途，比如用于商业的就是"com"，用于非营利性组织的就　　是"org"；第二部分是公司名，比如"tengxun"、"baidu"、"alibaba"；第三部分是你的项目名

**3、artifactId**

　　可以认为是Maven构建的项目名，比如你的项目中有子项目，就可以使用"项目名-子项目名"的命名方式

**4、version**

　　版本号，SNAPSHOT意为快照，说明该项目还在开发中，是不稳定的版本。在Maven中很重要的一点是，**groupId、artifactId、version三个元素生成了一个Maven项目的基本坐标**，这非常重要，我在使用和研究Maven的时候多次感受到了这点。

在上面的这些元素之外，还有一些元素，同样罗列一下：

**1、packing**

　　项目打包的类型，可以使jar、war、rar、ear、pom，默认是jar

**2、dependencies和dependency**

　　前者包含后者。前面说了，Maven的一个重要作用就是统一管理jar包，为了一个项目可以build或运行，项目中不可避免的，会依赖很多其他的jar包，在Maven中，这些依赖就被称为dependency。

　　说到这里，就有一个**本地仓库**和**远程仓库**的概念了。官方下载的本地仓库的配置在"%MAVEN\_HOME%\conf\settings.xml"里面，找一下"localRepository"就可以了；MyEclipse默认的本地仓库的地址在"{user.home}/.m2/repository"路径下，同样找一下"localRepository"就可以找到MyEclipse默认的本地仓库了。

　　本地仓库和远程仓库是这样的，**Maven工程首先会从本地仓库中获取jar包，当无法获取指定jar包时，本地仓库会从远程仓库（中央仓库）中下载jar包，并放入本地仓库以备将来使用**。

　　举个例子，比方说我的项目中用到了MyBatis，那么可以这么配置：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <dependencies>      <dependency>          <groupId>org.mybatis</groupId>          <artifactId>mybatis</artifactId>          <version>3.2.5</version>      </dependency>  </dependencies> |

    之前有说过groupId、artifactId、version唯一标识一个Maven项目，有了这三个元素，我们就可以去远程仓库下载MyBatis3.2.5.jar到本地仓库了。回想我们之前的做法，如果要MyBatis的jar包，发现没有，然后去网上下载一个，需要另外的jar包，然后去网上下载一个，但是有了Maven，就方便多了，只需要配置jar包对应的dependency依赖，Maven会自动帮助我们去远程仓库中下载jar包到本地仓库中。

**3、properties**

　　properties是用来定义一些配置属性的，例如project.build.sourceEncoding（项目构建源码编码方式），可以设置为UTF-8，防止中文乱码，也可定义相关构建版本号，便于日后统一升级。

**4、build**

　　build表示与构建相关的配置，比如build下有finalName，表示的就是最终构建之后的名称。

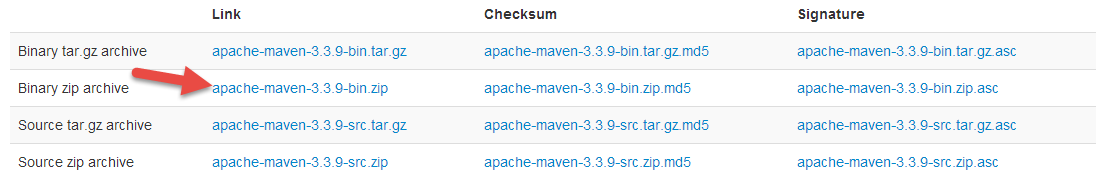
接着解释一下Maven的目录结构：

* main目录下是项目的主要代码，test目录下存放测试相关的代码
* 编译输出后的代码会放在target目录下
* src/main/java下存放Java代码，src/main/resources下存放配置文件
* 这里没有webapp，Web项目会有webapp目录，webapp下存放Web应用相关代码
* pom.xml是Maven项目的配置文件

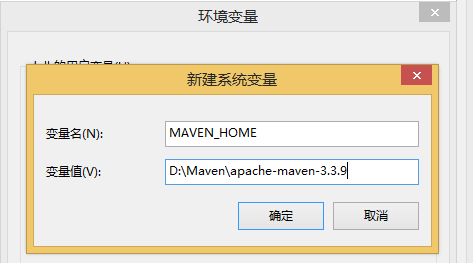
**三.Maven环境配置**

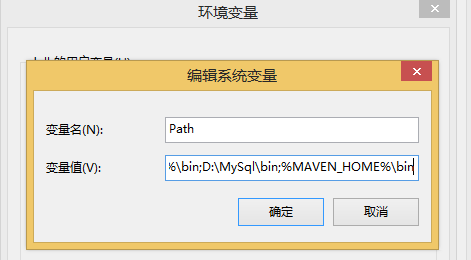
常用的开发工具Idea、MyEclipse里面都已经集成了Maven了，不过最好是从官网下一个配置到自己电脑里，开发工具里的可能有少许的Bug。

首先去Maven官网，下载Maven的包，地址为http://maven.apache.org/download.cgi，找到下面的部分，点击就可以下载了：

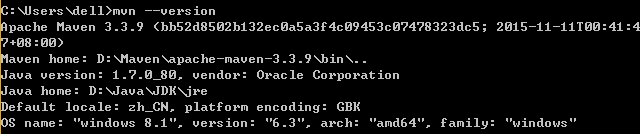


下载完解压，然后配置一下环境变量，和JDK的环境变量配置类似：





这样配置完就可以了，验证一下，windows+r打开命令窗口，输入"mvn --version"，如果有下面的内容这表示Maven配置OK：



**Maven入门使用常见问题**

**1、我从SVN上下载了一个以Maven构建的工程，下载完毕没有jar包怎么办？**

从资源库上下载下来的Maven工程，是没有jar包的，此时可以这么做：

（1）删除Maven工程，但是删除的时候不要删除本地工程里面的Maven工程，只删除MyEclipse中的Maven工程

（2）右键Import->Maven4MyEclipse->Existing Maven Projects，导入你的Maven工程，此时MyEclipse在构建工程的时候，假如你的本地仓库中没有dependcy中的jar包，便会去远程仓库下载jar包到本地仓库中。你的工程导入完毕之后，Library应该是这样的：



**2、重新下载Maven工程所依赖的jar包、导入jar包都需要触发Maven工程的build workspace，那么如何才能触发Maven工程的build workspace？**

找到一个pom.xml，随便修改一下，加一个空格、减一个空格什么的，ctrl+s保存一下，便会触发Maven工程的build workspace了

**3、本地仓库的目录结构是什么样子的？**

groupId、artifactId、version确定一个唯一的Maven，比方说我有一个mybatis的dependcy:

|  |  |
| --- | --- |
| 1  2  3  4  5 | <dependency>      <groupId>org.mybatis</groupId>      <artifactId>mybatis</artifactId>      <version>3.2.5</version>  </dependency> |

那么mybatis的jar包应该在%Maven仓库地址%\org\mybatis\mybatis\3.2.5\这一路径下，看到了吧，先groupId，再mybatis，最后version，每个名字都是一个文件夹的名字

**4、有些jar包在dependcy里面有配置，Import了Maven工程，下载完工程所依赖的jar包之后，发现本地仓库里面却没有，怎么办？**

可能是原先下载jar包的时候出了什么问题，从artifactId目录开始删除以下的所有文件/文件夹，然后触发一次Maven工程的build workspace就可以了

**5、本地仓库中确定已经有jar包了，工程里面却报错，说找不到jar包，该怎么办？**

应该有很多解决办法，目前解决的一种办法是，MyEclipse->Window->Preferences->搜索Maven->User Settings,Update Settings和Reindex点一下就好了。另外，可以尝试一下把本地Maven仓库内的jar包删除一下，然后重新build workspace，可能也可以。